


Striking gold with your XRootd caching proxy

Alja & Matevž Tadel

Talk outline:

1. Motivation, context
2. What was there, the story of what we did
3. How the thing works
4. What we already know should be done next

amraktadel@ucsd.edu, mtadel@ucsd.edu

Motivation & context

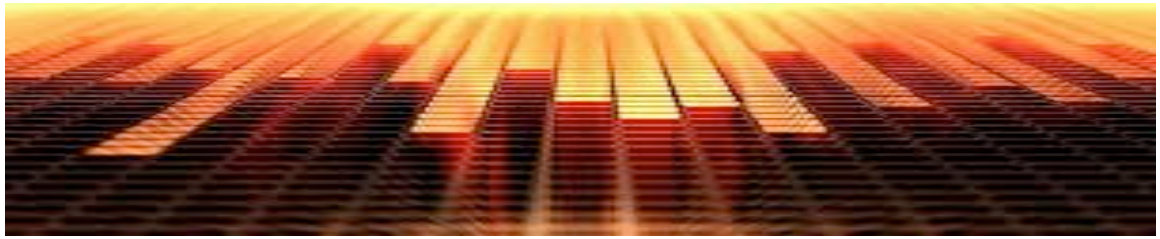
- Caching proxy was part of the AAA proposal
  - Planned for third year … and we started early and are finishing late
- AAA use-cases:
  - reduce latency, serve cases where a remote file is read multiple times (e.g. mixing, pileup, analysis)
  - T3 user analysis
  - now T2s seem even more promising
- Other VOs, esp. those with non-optimized IO
  - makes a lot of sense for OSG
  - one could use XrdPosix in client or XrdHttp on the proxy

A somewhat emotional interjection

- Caching proxy is a hard problem
  - One tries to makes the best out of a (few) machines:
    - saturate network interfaces,
    - operate disks near read/write limits (simultaneously),
    - make good use of buffers in RAM.
  - And people throw vector reads spanning whole 4GB files at you.
- Internet / hardware / kernel / XRootd … everything comes to bite you at some point.
  - When we started, it was all of the above together ☺

Striking gold?!?

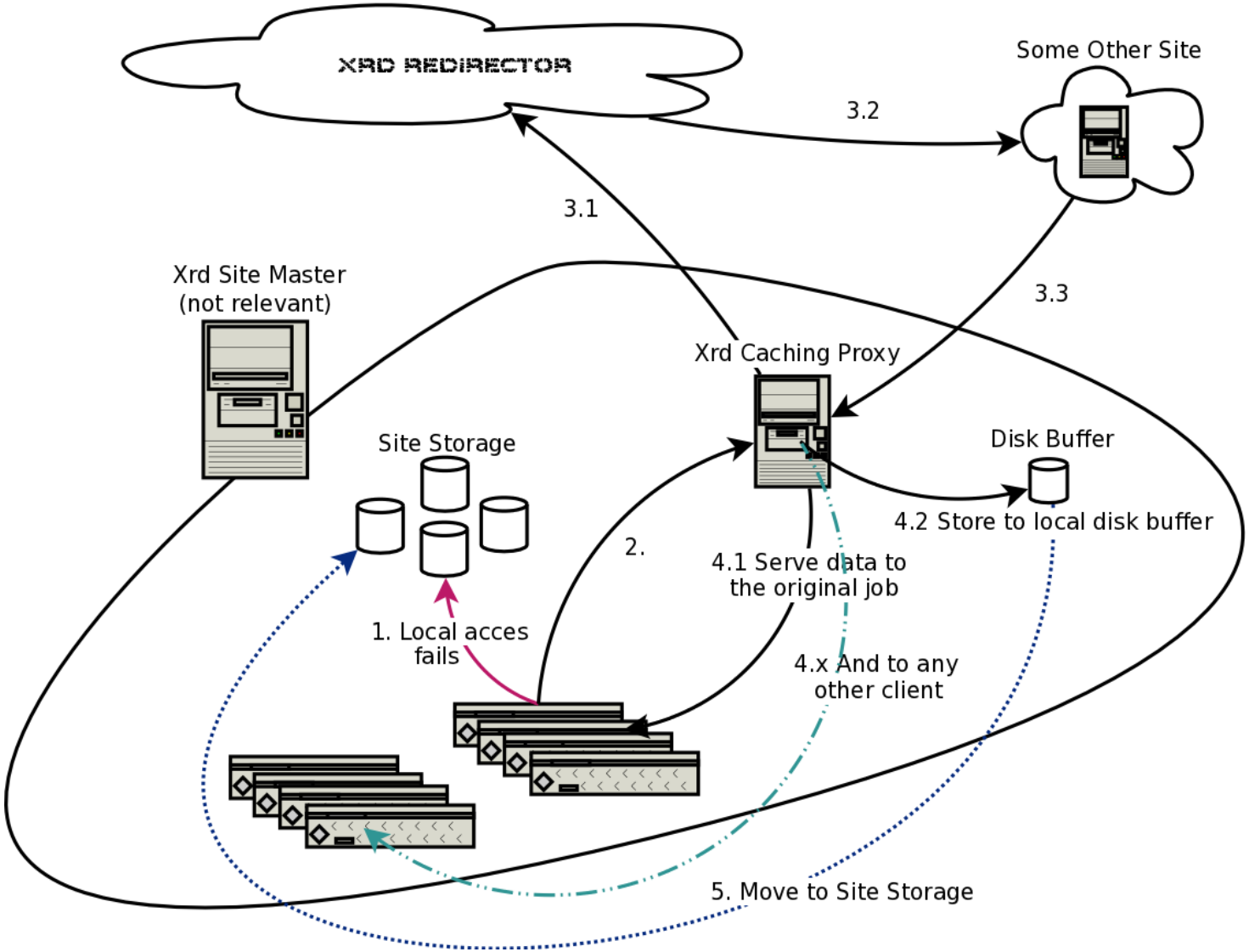
- At this point you've probably realized the title is, to some extent, ironic
  - effort was underestimated by a factor of 3;
  - we've never dealt with net/disk at saturation levels;
  - we were both new to XRootd internals;
  - Alja had little experience with multi-threaded code.
- So … it was fun … and now there is hope, too!



1. The original plan
2. What was already available
3. What's gone wrong and what helped to get around it
4. What's been done so far

THE STORY OF WHAT WE DID

The plan & a high level scheme



The plan II.

- Prefetching:
  - The proxy can go ahead and download parts of the file while client is processing its current hunk
  - This is stored to disk (along with "direct" requests)
- Storage healing:
  - If data was supposed to be on the cluster, we can inject it from the cache into local storage
  - What if only part of the data is missing?
    - Make proxy aware of segmentation, inject only missing blocks ➔ we have this 80% done for HDFS

What was already in XRootd

- Proxy service / cluster – libXrdPss.so
  - Implemented as an oss plugin using XrdPosix interface to forward requests to a proxy client.
  - The main purpose was to provide access into and out of private networks.
- In memory cache and read-ahead supported.
  - On the proxy level – specify block size and total cache size.
  - On the client level.

  See *ofs* documentation for details.

Connecting caching-proxy into XRootd

- XRootd allows to register a plugin that:
  - Is instantiated as a Cache object
  - This then gets called to instantiate a CacheIO object, one per open file.
  - CacheIO has to implement various Read calls
- Within this, one can implement all needed functionality for prefetching, writing to local disk, reading from it or from a remote source.

Basic design

- Read request coming into CacheIO are rounded to a block size (64 kB – 8 MB).
  - Data is returned to the client.
  - Block is also written to local disk (via a write queue).
  - Vector reads: serve what we have, pass through the rest
- A prefetching thread is started for each open file:
  - blocks are downloaded in order and also written to disk
  - prefetching continues for as long as the file is open or the whole file is downloaded.
- If the block is already available, data is read from disk.
- Basic cache purging is available (low/high water mark).

Problems along the road I.

- Network interfaces on oldish boxes can not saturate 1 Gbps both ways! Really.
  - Use *iperf* to measure performance.
- Disks/raids/lvms vary wildly in how they behave under heavy load.
  - Use *fio* to simulate desired loads and see where / how things break.
- Remote access is fun, too:
  - routes, problems on the way
  - remote cluster issues (which you can't know much about)
- All together, things can vary by orders of magnitude between a good and a bad day:

Always do basic measurements when something looks strange. Trust me on this ☺

Problems along the road II.

- MLSensor helped us a lot to see what is going on. All our plots are done with this.
- Issues with XRootd – the proxy layer was never subjected to the brutality we were inflicting on it.
  - We had to switch from XrdClient to XrdCl in the XrdPosix layer.
  - A couple of issues with locking – VTune captured all of them rather well.

1. Basic tests
2. Tests on an I/O node for 100 Gbps networks
3. Healing HDFS storage with hdfs-xrootd-fallback and a block-based proxy

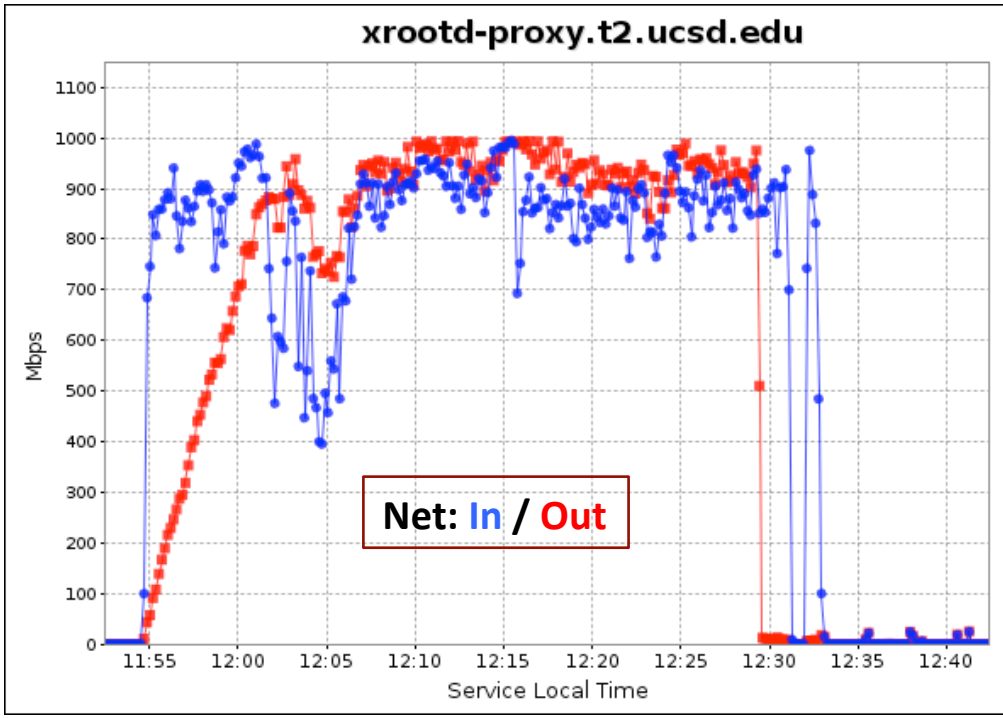
HOW WELL DOES IT WORK?

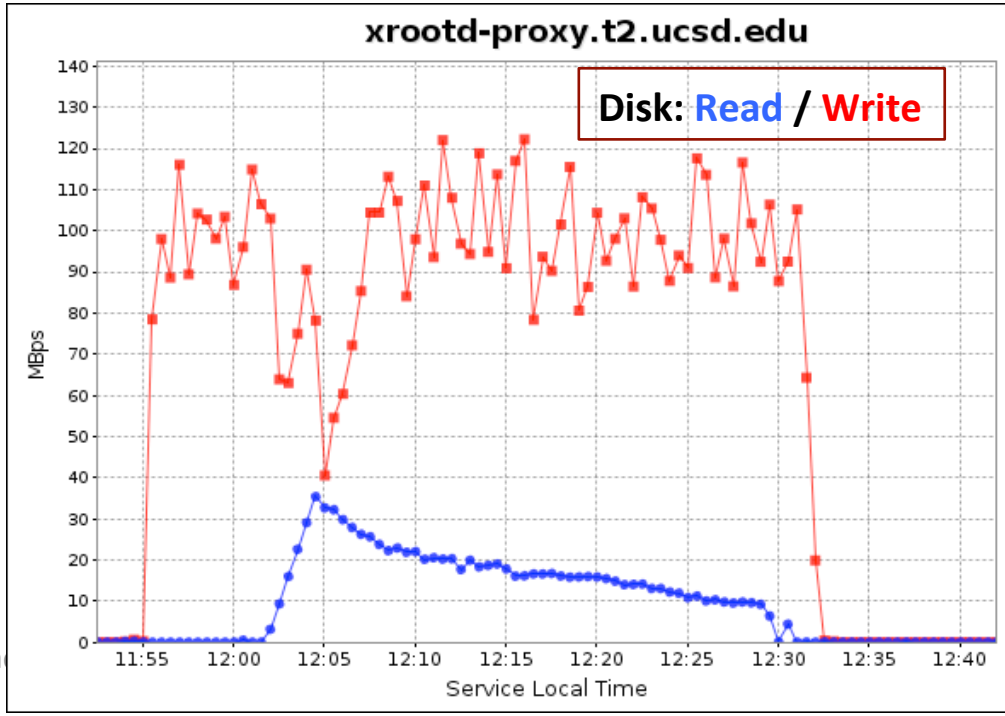
Standard test

- Standard job:
  - reads 2.5 MB every 10 seconds
  - *xrdfragcp –cms-job-sim* → sequential single reads
  - Run **a lot** (several hundred) of those against a site or the whole federation
    - we pick LFNs from detailed monitoring
- Then plot network / disk I/O rates
- Running on a "standard", 5 year old worker node:
  - 48 GB RAM, 4x2TB disks in a LVM volume
  - 2 x 1 Gbps (one on T2 network, the other on WAN)
- All tests run with 4MB block size, I think

600 jobs reading from UNL cache empty

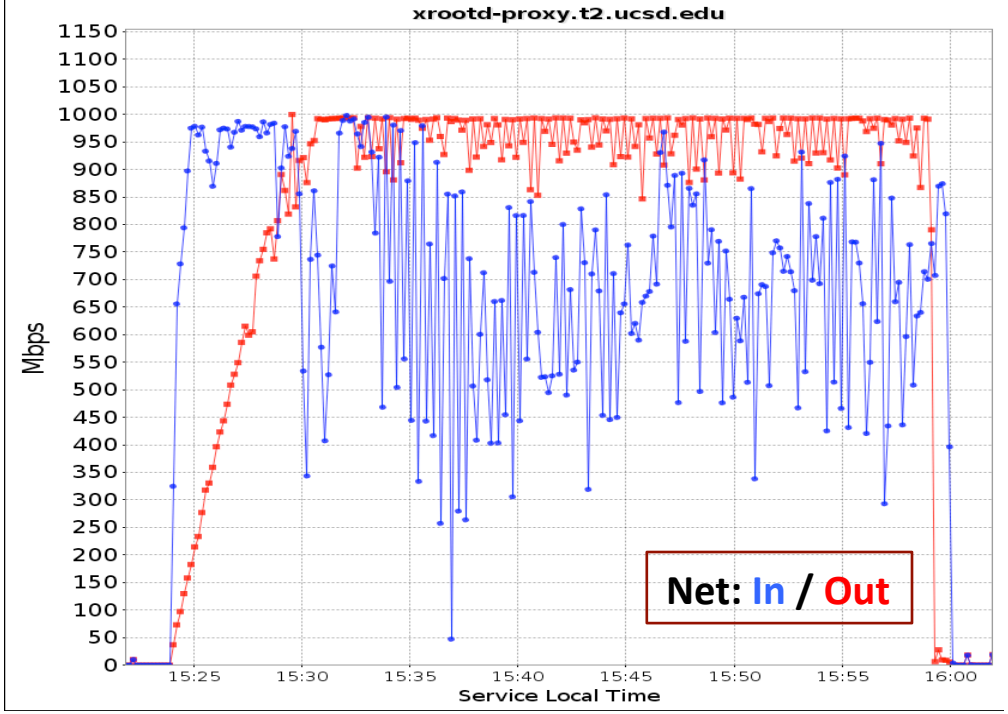
- Prefetch starts strong ☺
  - Disk swallows
- After 10’ disk read starts
  - write slows down
  - RAM allocated for prefetch gets used up
  - prefetch slows down, too
- Equilibrium is reached within minutes
- At the end, the jobs were killed forcefully, all at the same time
  - net out drops
  - write queue still needs to be flushed

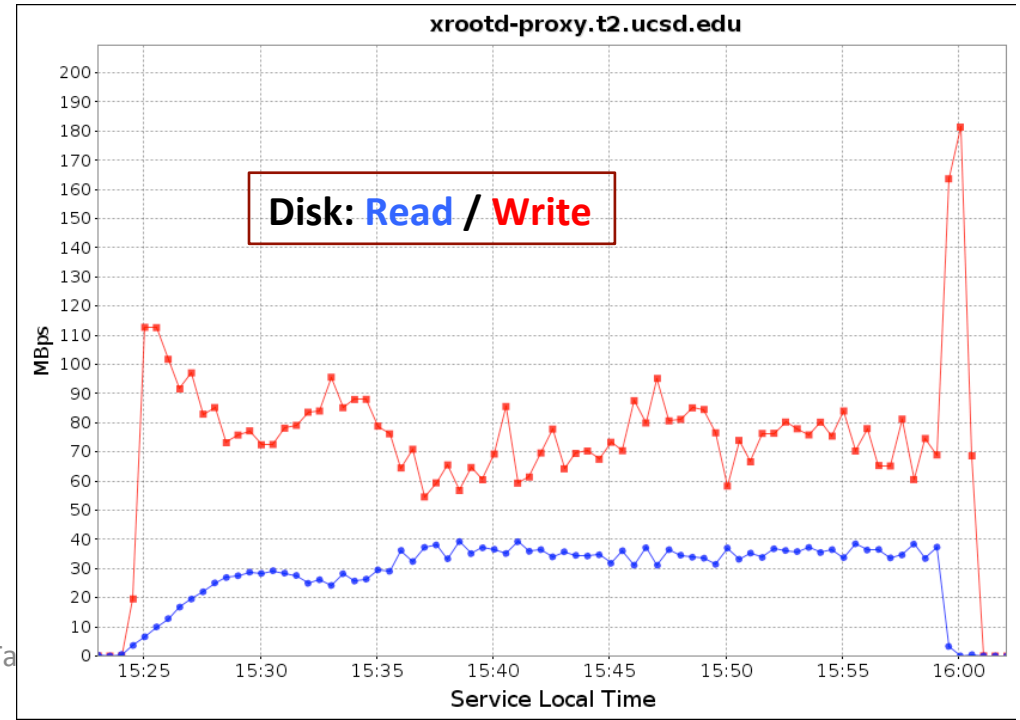




600 jobs reading from UNL cache 1/3 populated

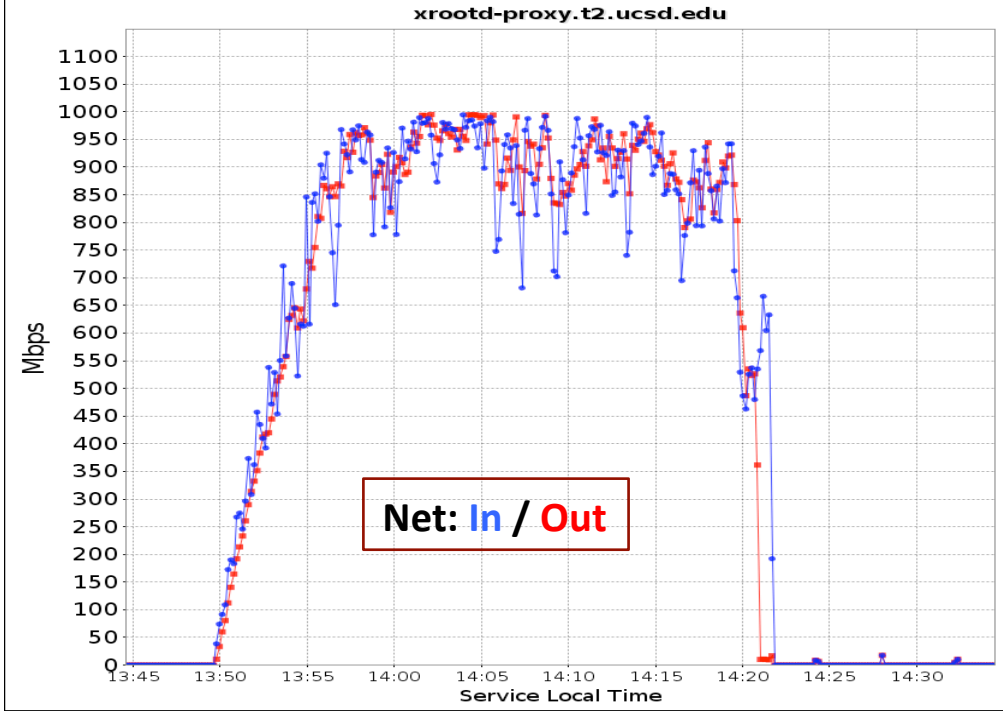
- Read starts right away
- Writes slower – disk overload
  - prefetch also slower

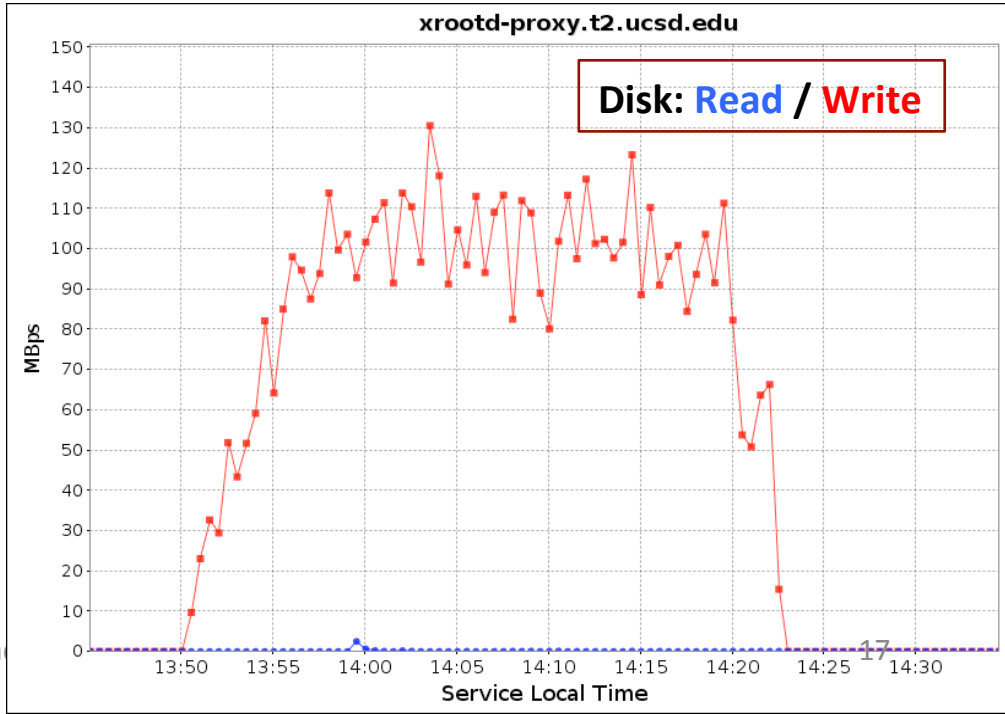




600 jobs reading from UNL no prefetching

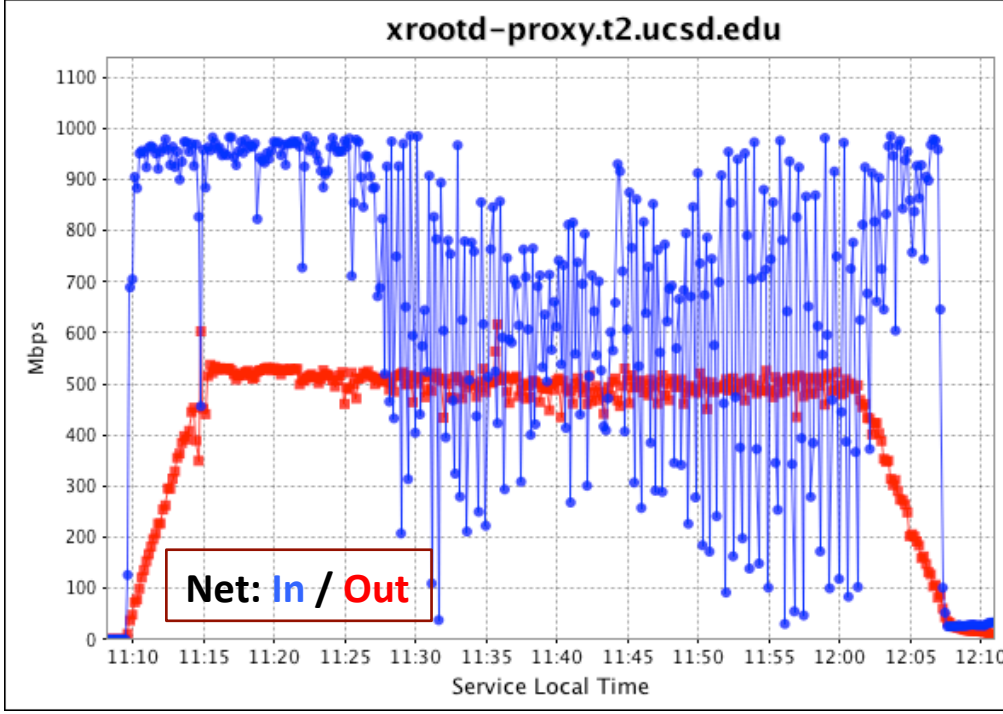
- Only requested data gets downloaded:
  - served to the client
  - and written to disk
- No disk reads

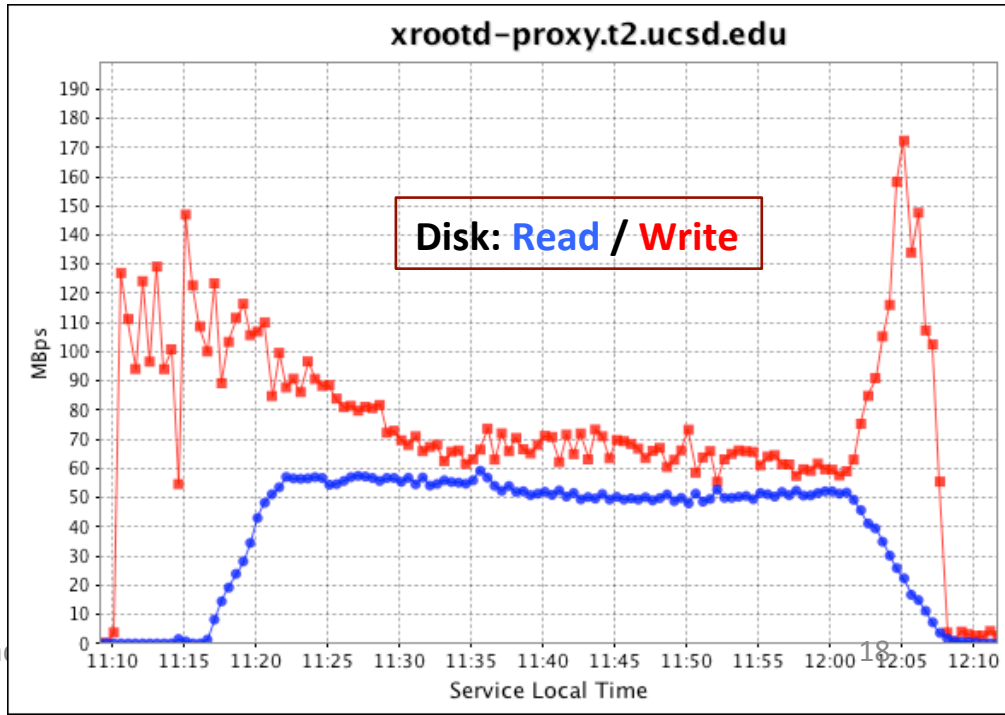




250 jobs reading from UNL

- This is ~50% of max load
- Prefetch charges ahead at 2 x speed
- In 10' all data is served from disk
  - which slows down writes
  - and things balance out around disk capabilities
- Jobs which were gradually added finish after an hour:
  - Reads drop gradually and write & prefetch pick up for remaining jobs.





PrismHEP – T. Hutton – 4/4/2014

Arista 7150-sw
meyer 5575

2 x 40G

Arista 7504
Prism-Core-Sw

HEP-6509-sw

10G fiber

Te1/4

Te9/9

10G copper

Gi6/48

1G

2 x 10G

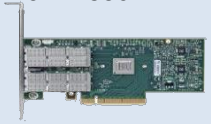
xrootd.t2.ucsd.edu

phiphi.t2.ucsd.edu

Mongo

9 X 256GB
510MB/sec

8 X 3TB
125MB/sec



2 x 10GbE

2 TB Cache
24TB Disk

40G

10G

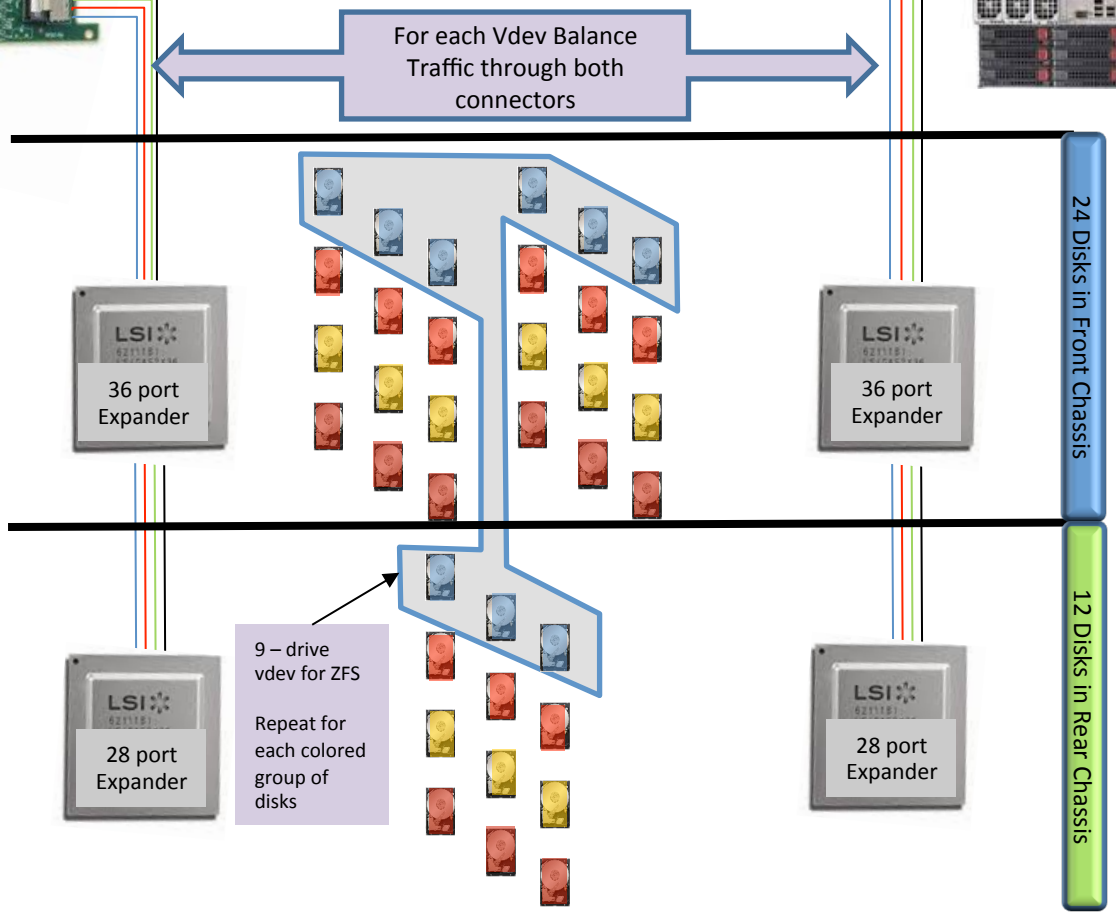
1G

108 TB disk, 64 GB RAM … ~18 k$

Data Appliance, 32GB

Phil Papadopoulos

VDEV Assignment for ZFS



- 36 Drives total in chassis
- 4 SAS-Lanes/Connector
- Drives are accessible from connector. Only one path to a drive is active
- ZFS vdev mapping:
  - All drives in a single vdev should be on same SAS lane/ channel (9 drives/ channel)
  - 4 vdevs total
- Multipath balancing – want traffic evenly split across both SAS connectors
  - Default multipath setup is to use first connector as primary

Phil Papadopoulos

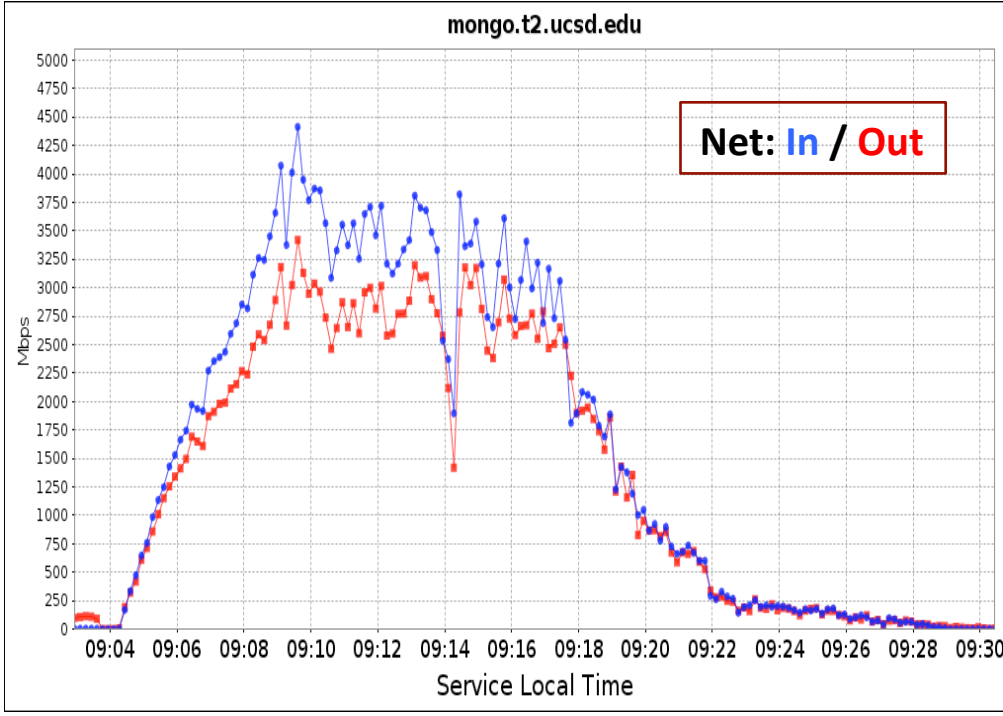
Mongo performance

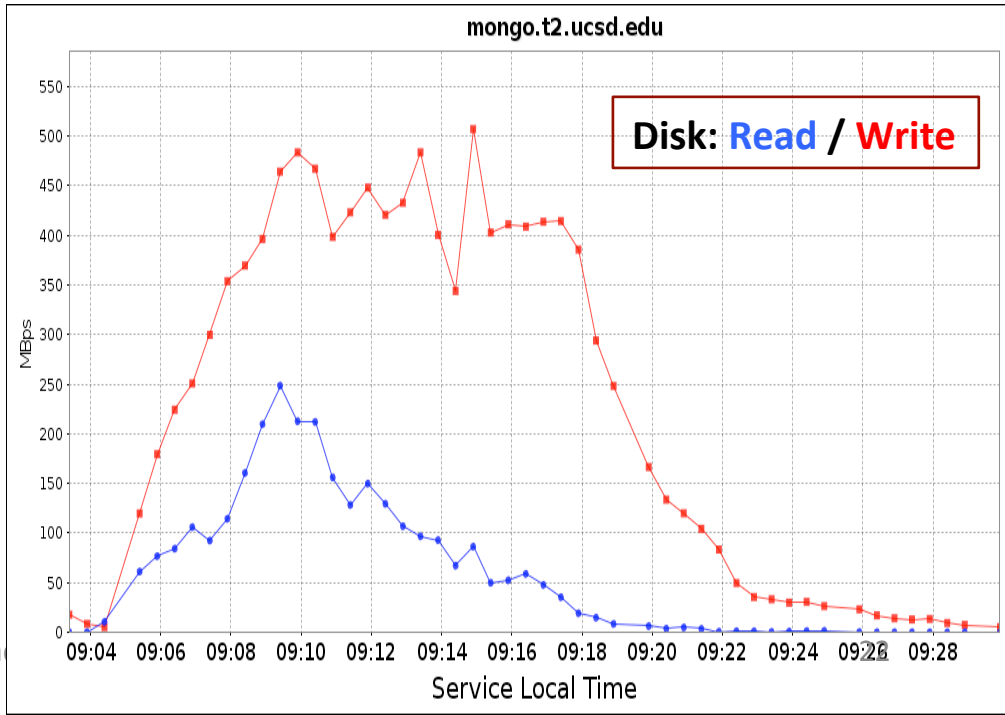
- Configure two zpools
  - Each with two 9-drive RaidZ2 vdevs (18 drives total)
  - Usable Storage in each pool is about 38 TB
  - Write performance for large streaming writes about 1.8GB/s
  - Read performance for large streaming reads about 1.6GB/sec
- Performance does NOT double when accessing both pools at the same time
  - Dual Read performance is about 2.8GB/sec (vs. 3.2GB/sec for doubling)
  - Dual Write performance is about 2.8GB/sec (vs. 3.6GB/sec for doubling)
  - Reason for fall-off is not understood.

This is also interesting for an "output buffer" machine!

FNAL → mongo, 3000 jobs

- The machine was just setup over the weekend – so this is preliminary.
- Yay, it works!
  - There was fear new things will crop up when going up x 4
- Route to FNAL is still limited to 5Gbps – so we manage to eat this up.
- There's clearly more *fun* ahead …





HDFS-XRootd fallback

- Talk by Jeff Dost this Monday in CMS section of OSG AHM
- In essence:
  - capture HDFS missing block exceptions
  - get the block via a local caching proxy
    - special version, block-size aware (passed in as URL arg)
    - prefetches and stores only the requested blocks
      - each block stored in a separate file (for easy re-injection into HDFS)
- Increases redundancy
  - also, can reduce replication for non-custodial data

1. Cache simulation based on IOV monitoring data
2. Modularization / simplification of the code:
   a. Async reads
   b. Address vector reads in a sensible manner
3. Options to make it really work in practice on near 100 Gbps network:
   a. Using few fat I/O nodes
   b. Using many standard nodes

WHAT WE ALREADY KNOW WE SHOULD BE DOING NEXT

Caching-proxy simulation

- Full timeline of read-requests from IOV → we can simulate the operation of a caching-proxy:
  - Same 190k AOD remote access data sample used
  - Parameters: block-size, prefetching rate
  - Questions:
    - How much data was served from the cache?
    - How much have we over-read the file?
      This changes with job progress!
    - How many trips to remote server have we saved?
- We just started playing with this:
  - Vector-read pass-through is not simulated
  - Extend to simulate a set of jobs
  - Take into account access (for locally run jobs)

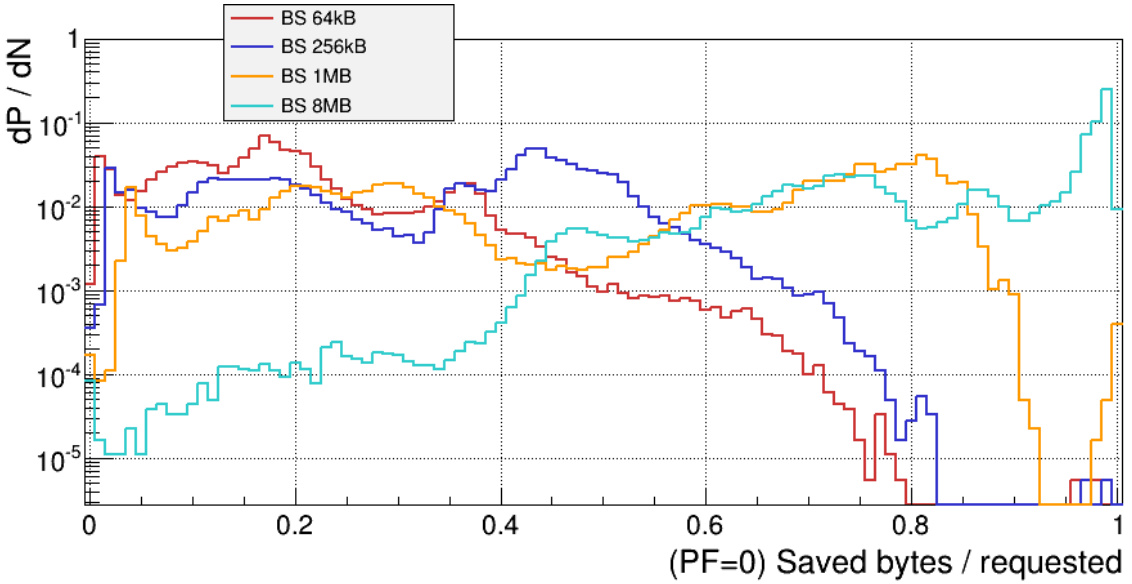
Caching-proxy simulation
Data served from cache

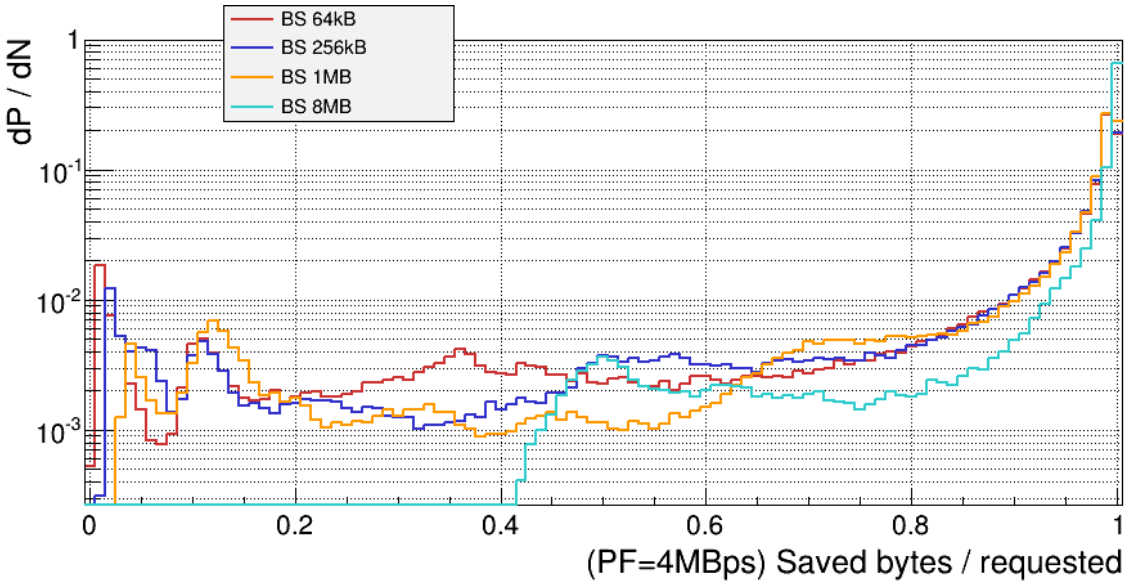
No prefetch:

- The larger the block size, more we have in cache

With prefetch on this is even more pronounced:

- jobs read at ~250 kB/s
- so this is about x16
- + the block-size effect



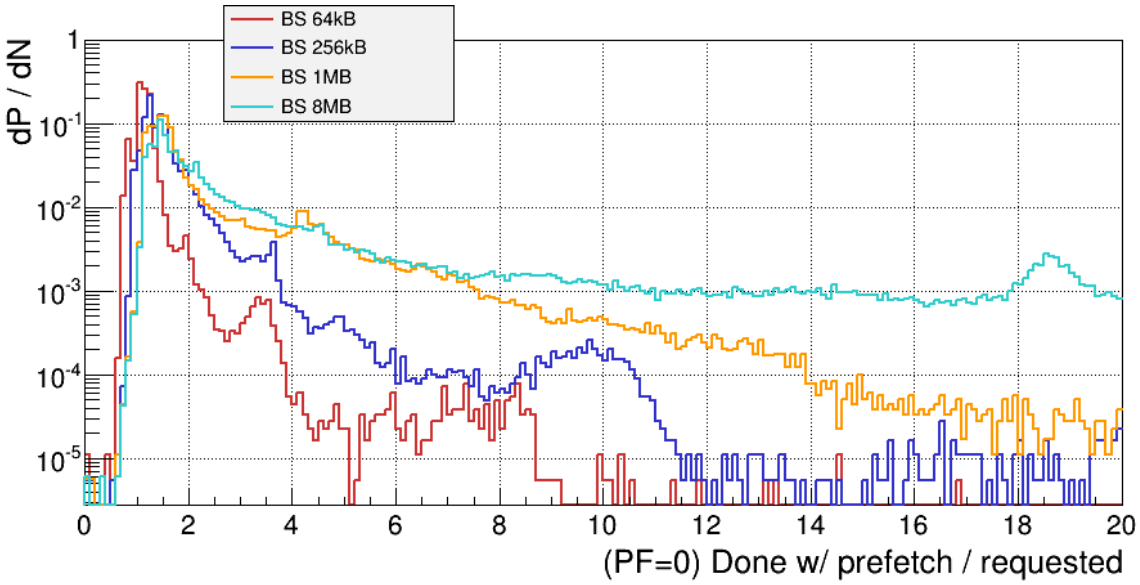


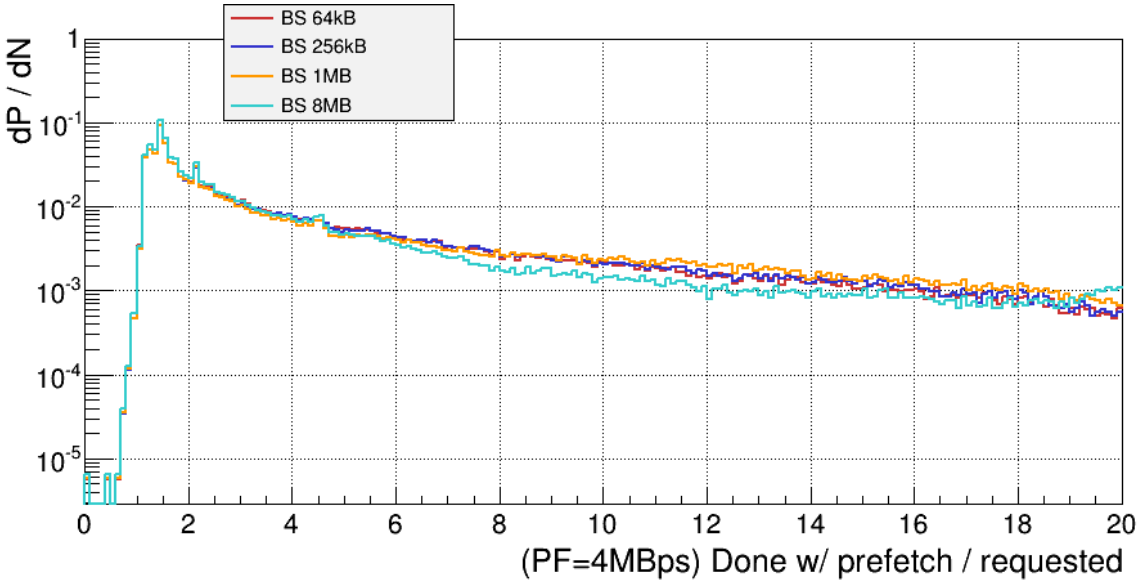
Caching-proxy simulation
Data fetched / requested

Of course, we have to read more than we need …

About 1.5 times more, most probable value

Obviously, cases where a small part of file is read generate the long tail.



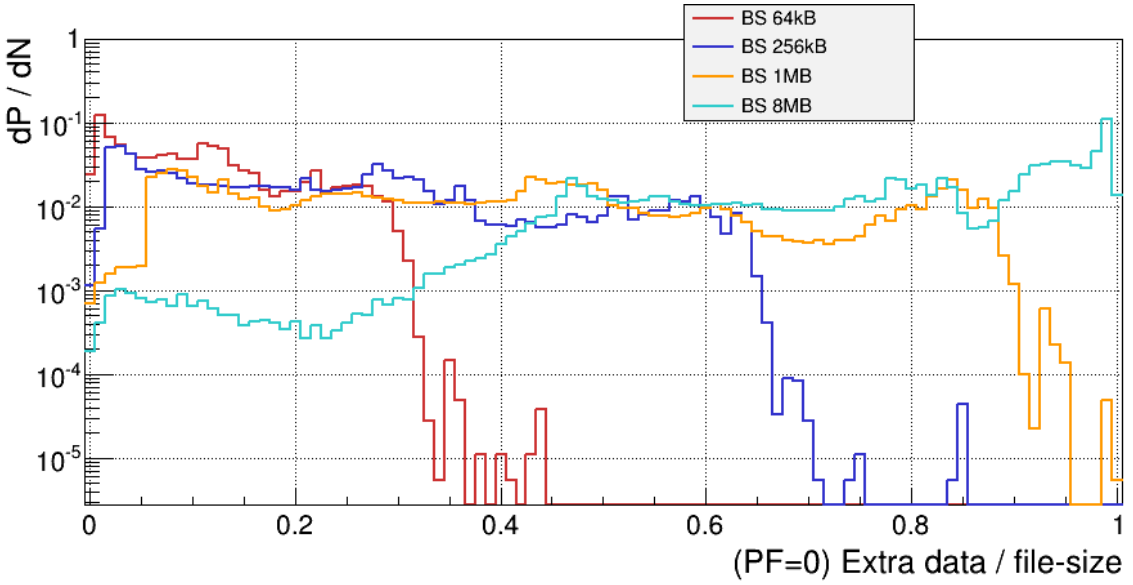


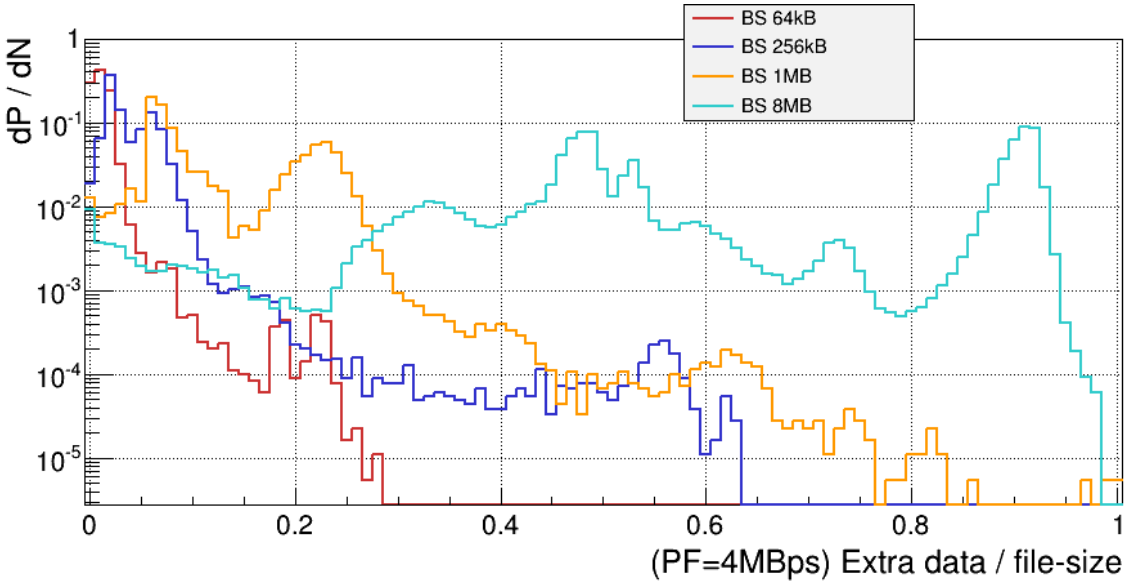
Caching-proxy simulation Extra data / file size

This is extra data from read requests only (no prefetch), summed up throughout the job.

- Part of this later get used (in “served from cache”)

This means we read too much in the beginning (of course).

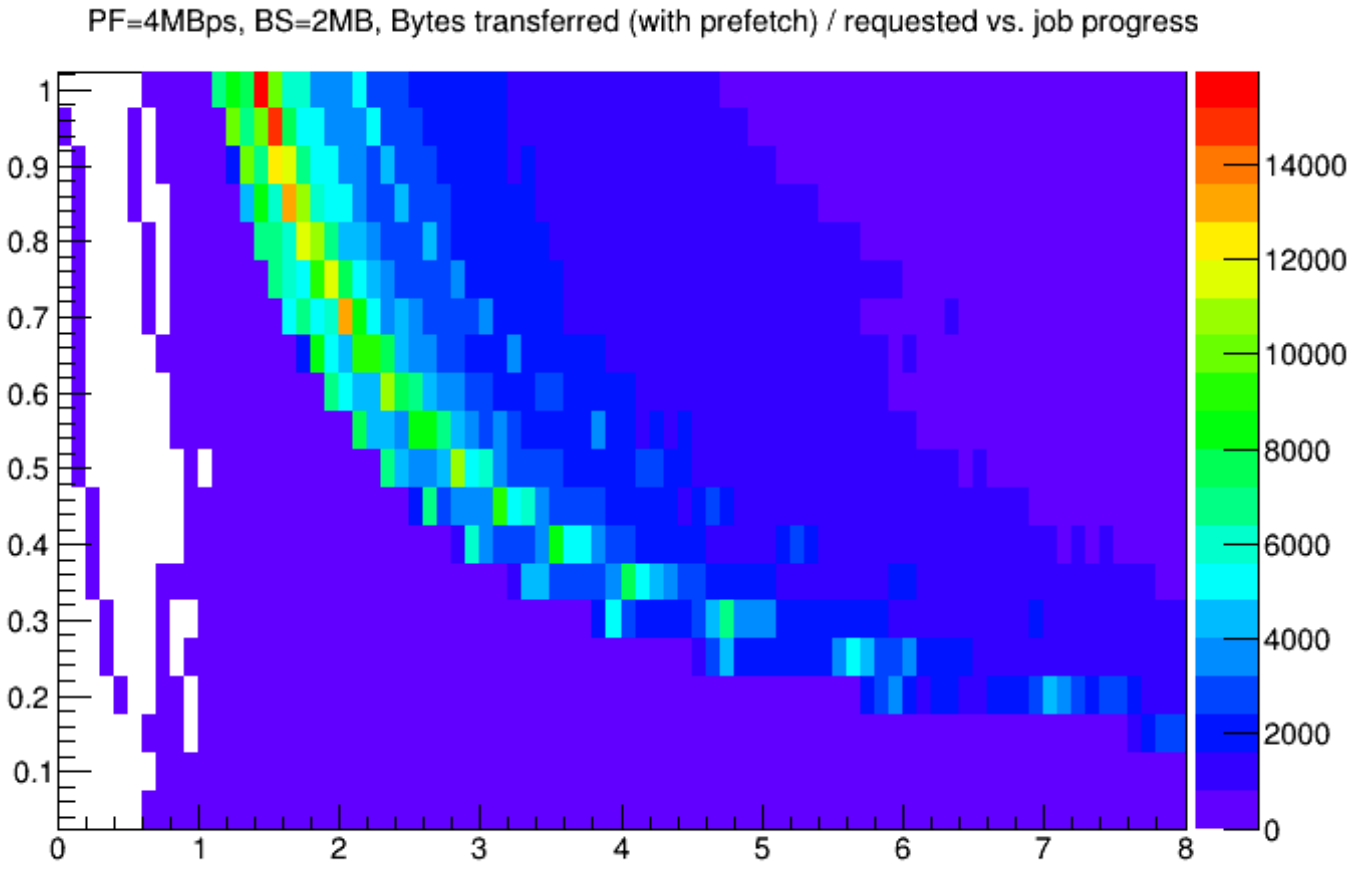




Caching-proxy simulation
Overread vs. time

All over-read, including prefetch:

- at 20% progress we've read 7x the required data
- and then end up at 1.5



Improve guts of the code

- Use asynchronous reads
  - reduce number of threads, be more nimble for block juggling
- Think how to support vector reads better
  - Now: serve what we have, forward the rest
  - The problem:
    - several hundred requests, each about 10 – 20 kB
    - offsets 1 – 10 MB, total extent up to 1 GB
  - One could choose a smaller block-size –kill the disk!
  - Or be more aggressive with prefetching – 100 Gbps networks!
  - There is no silver bullet … depends on needs / affordability

  We have to find a way to support different strategies.
- Multi-source reading / switching during access
  - Can happen behind the back of the actual client!

How to provision large installations?

- Caching-proxy cluster
  1. few mongo like machines; or
  2. interspersed in the cluster so that individual load is small

  We started playing with this …
- Operate fraction of Tier-2 storage as a cache?
  - Tempting, as there is no data-placement, it happens on-demand, only data that is actually needed.
  - Job-placement easier – we don't have to know where data is, only where it might have been.
  - Still, it makes sense to orchestrate site-targets for data-sets.
- And here one might strike the gold vein …

Conclusion

- Le Caching-proxy v1 est arrivé!
  - Available in xrootd-4.0.0
- This is not the final code but:
  - We came a long way, can saturate NICs and disks, both ways. Oh, and use all your RAM ☺
  - All the pieces are on the table: testing setups, analysis / simulation tools.
- Who wants to try it out? Let's talk …
- A possible solution for non-HEP VOs in OSG